Object-Oriented Systems
-------------------------

There are lots of object-oriented things:

Programming languages (Java, C++)
Databases
Object models (UML)

Meilir Page-Jones
Fundamentals of Object-Oriented Design in UML
Addison Wesley
2000

UML = Unified Modeling Language

Now go to (4). Skip (1), (2), (3).

(1) Problems with Relational Modeling: Problems with Modeling.
Try to model something simple as MEN and WOMEN.

Solution 1: 1 Table.

Problems: Men could be pregnant. Schema implies it.
Data easily wrong. Many Null values.

Solution 2: 2 Tables. Problem: Can't easily compute simple
things like average age. Big amount
of duplication in schemas (age, name, address).

Solution 3: 3 Tables: PERSON, MAN, WOMAN. Problems:
Need foreign keys. Need joins. Psychologically, we want
all information on one object at one place.

For data modeling, the relational model is not ideal!
OODB model tries to solve these problems.

(2) Technical problems of the relational databases:

Due to normalization, considerable numbers
of expensive joins are necessary.
Information about one real world object is
spread out over many tables.

(3) New (in the 80s) Demands on relational databases:

Version information (Computer Aided Design
= CAD, Objects appear in many versions)

Long duration transactions ("atomic units")
are necessary in CAD.

Grouping (clustering) of related objects
(parts of an object) is desirable.

Support for group activities is needed.
An object might have been modified by somebody else or might be "unavailable".

Schema evolution is the rule not the exception. (That means, the schema keeps changing.)

Complex mathematical evaluation must be supported ("air flow on the wing").

Protection (security) on an object level must be supported ("keep all information on the FYZ fighter plane secret").

It was the vision of the OODB community that OODBs are better in supporting some or all of the new demands.

-------------------------

(4) Object-Oriented Modeling with UML (Unified Modeling Language)
[Grady Booch, Ivar Jacobson and Jim Rumbaugh]

Look here. There are now 14 diagrams.

Before you start programming you make an object-oriented model of your domain.
Object-Oriented modeling uses a graphical language, typically a version of UML.

Most earlier programming approaches concentrated on functions.

Objects

The big idea of object-oriented programming was this: We are writing programs about real world things. Cars, students, employees, tax payers, DVDs, library book, etc. So there should be one "thing" in the program that models one "thing" in the real world. These "things" are called "objects".

Classes

Next we notice that there are many "things" that have a lot in common. All cars have certain things in common. All people have certain things in common. Thus we create our objects from "templates" that capture what is common. Such a template is called a "class." Graphically these are boxes.
The class name is at the top inside the box.

http://www.uwgb.edu/breznayp/images/uml.jpg
(Objects are also shown as box, with the class name underlined. Objects belonging to a class are called Instances of the class.)

Attributes

Now we know that all real world objects have certain attributes: Cars have colors. Students have grades. Employees have salaries. Library books have authors. So, we need to model real world attributes of "things" as "attributes of objects". Graphically these are text lines in the box below the class name. These lines consist of attribute name and data type. Attributes have simple data values for example number or string.

Operations

There are certain things that can be done with objects of a specific class. Operation implementations for objects from a specific class are called "methods." Basically these are functions where the first argument must be an object of the class and is written BEFORE the function name. Graphically operations are text lines in the box at the bottom, below a separator line. Such a text line consists of the operation header.

Relationships

Next we notice that real world things "relate" to each other. Thus a person may have borrowed a book from a library. Of course he may have several books from several libraries.

A person may own a car, be married to another person, etc. We will call each such connection between two classes a relationship.

Note that this approach is a simplification of the usual UML approach based on "associations." See here for some criticism of the UML approach:

Relationships appear as arrows from one box to the other box with a label on the arrow, expressing the name of the relationship.
In the Box diagram, relationships may appear just like attributes, but the "data type" is the name of another class. Still draw the arrow, but without label.

Class Hierarchies

Next we notice that some classes are more general than other classes. Thus, the "specialized" class has all the attributes, and operations of the general class plus some additional attributes and operations.

Example: Vehicle is a more general class than car, truck, airplane, ship.
Car is a more general class than SUV, Sedan, sports-car, etc.

Instead of saying that you need
- a license to drive a sports-car
- a license to drive an SUV
- a license to drive a truck
- a license to drive an airplane
etc. etc.

We say you need a license to operate any vehicle.
Then we say that a car IS-A vehicle, a truck IS-A vehicle, etc.

Graphically, we express this fact by drawing a triangle hollow head arrow from the specialized class to the general class. This arrow is called IS-A link or subclass link.

Now that we know that the subclass (specialized class) has all the information of the general class we don't need to write the general information !!! This is Software Reuse!

We try hard to draw the general classes ABOVE the specialized classes.

Together, all the classes and IS-A links (= SUBCLASS links) form a TREE or a DIRECTED ACYCLIC GRAPH (=DAG).
That means, the SUBCLASS LINKS never form a cycle. (You can't get back to the starting point.)

In UML arrow heads are shared. They may be marked as disjoint (no object in two subclasses) and complete (all objects in at least one subclass). Also, we are *trying* to keep lines horizontal/vertical.
Parts: Composition and Aggregation

Composition is a whole-part relationship. It is graphically marked by a full diamond head arrow at the "whole."

E.g. Axe has parts handle and head.

Aggregation is a set-membership relationship. It is graphically marked by a hollow diamond head at the "whole."

E.g. Committee has as parts people.

This maps very cleanly to C++/Java

Class -> Class of Java or C++
Attribute -> Data Member
Relationship -> Pointer Data Member in C++ or Reference in Java
Object -> Object (Instance) created with "new"
Operations -> Member functions in C++
           "non-static" functions in Java
IS-A -> extends key-word of Java subclass of C++

For more on UML check here:
http://agile.csc.ncsu.edu/SEMaterials/UMLOverview.pdf

(5) Where to look things about DB research up:

DBLP Database and Logic Programming
Archive on the Web:
http://www.informatik.uni-trier.de/~ley/db/

(6) Micro History of OO ideas:

Object-Oriented Programming languages:

SIMULA (Dahl & Nygaard, 1966)
SmallTalk   (Adele Goldberg & David Robson)
C++ (Bjarne Stroustrup, 1986)

Object-Oriented Databases were invented here:

A. James Baroody Jr., David J. DeWitt: An Object-Oriented
The simple idea was to take the SmallTalk Object-Oriented Language and make it persistent.

Persistence is the most fundamental feature of databases. Something (data, procedure) is persistent if it is created by a program or person, but is still there after the program has finished.

That means, if your program creates objects, next time you log in, they are still there. Like tables are still there if you log out and log back in.

Today few companies that make OODBs exist.

Versant disappeared... and then reappeared, see email by Dingersoll (David Ingersoll).

Object Store
http://www.objectstore.com/product
now markets itself primarily as an in-memory database system.

Ontos does not seem to exist anymore.
It went "Chapter 7" in 2007.
http://leagle.com/decision/2007905478F3d427_1904.xml/IN%20RE%20ONTOS,%20INC.

The following is based on the book Object-Oriented Database Systems by Elisa Bertino and Lorenzo Martino Addison Wesley. 1993.

(7) Features of Object-Oriented Databases:

1. Unique Object Identifier (OID)

An Object ID is a unique number within a database. Every object has a different OID. OIDs "never" change.
2. Complex Objects with Attributes

Arrays of objects are also considered objects. The same applies to sets and list of objects. An operation defined for the complex object could be "count".

3. Encapsulation

Two aspects:

1: Data is hidden behind a wall and not accessible. For instance private data in C++.

2: Data and the functions that manipulate the data are bundled together into the object. So the functions that operate on an object are also behind the wall. At least in principle. (In practice we don't copy the code over and over.) Typically we make the functions "public" so that they can be called from outside of the object. (Here: function=methods).

OODBs cannot support encapsulation completely. It is impossible to query data if you don't know the names of the data fields! With perfect encapsulation you would not know the data fields!

4. Classes

The hierarchy of classes is considered the "schema" of the object-oriented database.

Diagram ---> Schema

5. Inheritance

Inheritance means that attributes/relationships/methods defined at a general class automatically exist at the specialized class. This gives you a "software reuse" effect. Imagine the attributes are "trickling down" along the IS-A arrows (against the direction of the arrow head).

Multiple parents (multiple inheritance) causes practical problems. Example: If you inherit the same
attribute name with two different types from two different classes, which one should you use?

Restrictive solution (C++): Prohibit inheriting the same thing from two parents. Make it a compiler error.

JAVA: Doesn't allow multiple inheritance at all. (But "interfaces" provide some of the functionality of multiple inheritance.)

6. Overloading, Overriding, Late Binding Polymorphism

Polymorphism:

4 + 5
4.7 + 6.2

When you add two numbers with +, the compiler decides whether to use INT addition or FLOAT addition, which are different machine language instructions!!!

If the actual code of a function (or an operator like +) depends on the type of its arguments, this is called POLYMORPHISM.

Overriding:

In OODB you can define a function high up in the class hierarchy. This will be inherited down.

Overriding means, you can define a function of the same name at a lower class, and that one will be used, not the inherited one.

In other words, what the function actually DOES depends on the class of the object the function gets as argument!! (It's polymorphic!)

That means, the actual code is only known at runtime. That is why it is called late binding. In "normal" programs the code is known at compile time already!

Example of overriding:
Assume a hierarchy: PRODUCT, with 2 children: FOOD, CIGARETTES.

Variable: thing_I_buy gets assigned an object.
thing_I_buy.compute_tax();

Now, for any objects of the class PRODUCT the tax is computed by adding 7% to the price. However, for objects of the class FOOD, there is no sales tax in NJ. For objects of the class CIGARETTES there is a tax of $2.70 on every pack of cigarettes.


So, you have three different methods, all called "compute_tax" at three different classes. The methods at the lower classes override the method at the parent class.

Overriding: Same Function Name, Same Argument at Different Classes [at different hierarchy levels]

Overloading:

Same Function Name, Different Arguments, at the Same Class

Overloading means that the same function name defined at the SAME class does different things depending on the number or types of the arguments!

thing_I_buy.compute_tax('Elizabeth');

In Elizabeth there is a special business development district. The tax rate there is only 3.5%. So, we could add a parameter LOCATION to compute_tax. If that parameter is empty, the tax rate is 7%. If we supply a parameter: a city, if that city is Elisabeth, then the tax would be 3.5%.

compute_tax() does something different from compute_tax('SOME ARGUMENT')

(8) Basic Idea of Object-Relational Databases
-----------------------------------------------
Most OODB companies ran into problems.

See Michael Stonebraker's book.
Object-Relational DBMSs - The next great wave (1996).

Oracle supports these extensions and we will discuss them.

The main construct is still the TABLE.
(Like relational, unlike Object-Oriented.)
However, it is possible to create objects and to place one object into one column of one row (a field) of a table.

In the relational model only "primitive" data types such as number, string or date may be placed in one column of one row of a table. 1NF (First Normal Form).

Object-relational databases are NFNF (Non-First Normal Form).

Methods (operations) may be defined for objects in the object-relational model. Thus, this is like the OO model. The relational model has no code associated.

As a reminder, in the OO model, there are no tables at all.

<table>
<thead>
<tr>
<th>Relational</th>
<th>Object-Oriented</th>
<th>Object-Relational</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tables only</td>
<td>Objects only</td>
<td>Objects and data in tables</td>
</tr>
<tr>
<td>No methods</td>
<td>Methods</td>
<td>Methods</td>
</tr>
<tr>
<td>Table headers</td>
<td>Class Hierarchy</td>
<td>Table headers and classes</td>
</tr>
</tbody>
</table>